Instituto Politécnico Nacional

Escuela Superior de Cómputo

Estructuras de Datos

Práctica no. 1:

Evaluación de expresiones fijas

Profesor: Edgardo Adrián Franco Martínez

Integrantes:

* Calva Hernández José Manuel
* Luis Carlos Ruiz
* Daniel González Núñez

Grupo: 1CM7

**Introducción**

La Pila es una estructura de datos que almacena y recupera sus elementos atendiendo a un estricto orden. Las pilas se conocen también como estructuras **LIFO** (Last-in, First-Out, último en entrar primero en salir).

Una **pila** (stack) es una colección ordenada de elementos a los que sólo se puede acceder por un único lugar o extremo. Los elementos de la pila se añaden o quitan (borran) de la misma sólo por su parte superior, la **cima** de la pila.

Las operaciones usuales en la pila son Insertar y Quitar. La operación Insertar (push) añade un elemento en la cima de la pila y la operación Quitar (pop) elimina o saca un elemento de la pila. [2]

Una pila puede estar vacía o llena (en la representación con un array, si se ha llegado al último elemento). Si un programa intenta sacar un elemento de una pila vacía, se producirá un error, una excepción, debido a que esa operación es imposible; esta situación se denomina **desbordamiento negativo** (underflow). Por el contrario, si un programa intenta poner un elemento en una pila llena se produce un error, una excepción, de **desbordamiento** (overflow) o rebosamiento. [1]

**Planteamiento del problema**

Con la implementación del TAD Pila en C, (estática y dinámica) implementar un programa que valide y evalué una expresión infija.

Ejemplo: A\*((B+D)/C)+E^A

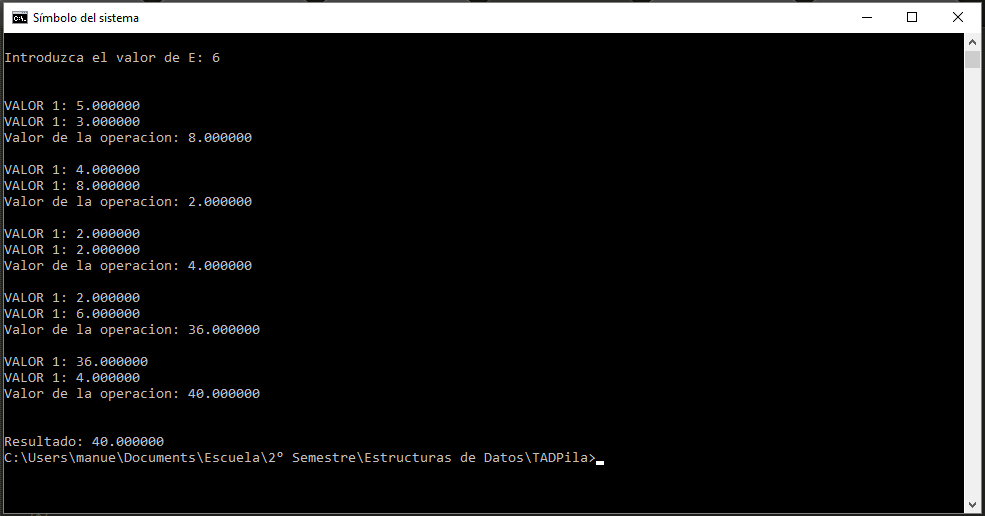
Considerar la precedencia de operadores siguiente:

1. Términos entre paréntesis ()
2. Potencia ^
3. Multiplicación y división \* /
4. Suma y resta + -
5. Evaluación de paréntesis escritos correctamente: El programa mostrara el resultado de la revisión de paréntesis
6. Conversión a posfijo: El programa mostrará el resultado de pasar la expresión infija a posfija.
7. Evaluación de la expresión posfija: El programa mostrará el resultado de la evaluación de la expresión, para ello solicitara el valor flotante de cada una de las variables (A-Z) que contiene la expresión.

**Diseño y funcionamiento de la solución** (Descripción de la abstracción del problema y su solución, apoyándose de diagramas y figuras en un lenguaje claro) \*Descripción del funcionamiento de los algoritmos que se apoyan del TAD pila.

**Implementación de la solución** (Según la solución diseñada como se implemento en el lenguaje de programación)

**Funcionamiento** (Verificación de la solución, pruebas y resultados de salida \*Pantallazos)



**Errores detectados** (Si existe algún error detectado, el cuál no fue posible resolver o se desconoce el motivo y solo ocurre con ciertas condiciones es necesario describirlo)

De momento no hemos encontrado un error como tal, sin embargo, existen restricciones que no comprobamos y que provocan que el programa se cierra, por ejemplo, a la hora de usar minúsculas o números al introducir la cadena de texto, o cuando se introduce algún otro carácter especial que no sean los que usamos como operadores.

**Posibles mejoras** (Describir posibles disminuciones de código en la implementación o otras posibles soluciones)

La primera mejora que notamos como posible, es la implementación para una cadena de texto que incluyera letras minúsculas, ya sea para convertirlas a su mayúscula, o para usarlas como una ampliación a las variables posibles para usarse.

La segunda, es una visualización del manejo de las pilas, para que sea más didáctico respecto al usuario.

En la parte del código, se puede reducir el uso de pilas a sólo dos, sin embargo, decidimos dejar tantas pilas para evitar confusiones con respecto a las funciones que ocupan las diferentes pilas.

**Conclusiones** (Por cada integrante del equipo)

**Anexo (Códigos fuente \*con colores e instrucciones de compilación)**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <math.h>  
#include "TADPilaDin.c"

void checar(pila \*s,char c[]){  
 elemento e;   
 int i;  
 for(i=0; i < strlen(c) ;i++){  
 if(c[i] == '('){  
 e.c='(';  
 Push(s,e);  
 }  
 else{  
 if(c[i] == ')'){  
 if(Empty(s)){  
 printf("\nError: Se cerro un parentesis sin abrir otro");  
 exit(1);  
 }  
 else  
 e=Pop(s);   
 }  
 }  
 }  
 if(Empty(s))  
 printf("\nNumero correcto de parentesis");  
 else{  
 printf("\nError en la expresion");  
 exit(1);  
 }  
return ;  
}

void cambio(char expre,pila\* s,pila\* guarda){  
 elemento e;  
 e=Pop(s);  
 printf("%c",e.c);  
 Push(guarda,e);  
 e.c=expre;  
 Push(s,e);  
 return ;  
}

void orden(pila\* guarda,pila\* inver,float\* valores){  
 elemento e;  
 while(!Empty(guarda)){  
 e=Pop(guarda);  
 if(e.c > 64 && e.c < 92)  
 e.n=valores[e.c-65];  
 Push(inver,e);  
 }  
 return ;  
}

void eleccion(elemento e,pila\* sum){   
 elemento r;  
 float x1=Pop(sum).n;  
 printf("\nVALOR 1: %f",x1);  
 float x2=Pop(sum).n;  
 printf("\nVALOR 1: %f",x2);  
 if(e.c == '+'){  
 r.n= x2+x1;  
 Push(sum,r);  
 }  
 if(e.c == '-'){  
 r.n= x2-x1;  
 Push(sum,r);  
 }  
 if(e.c == '\*'){  
 r.n= x2\*x1;  
 Push(sum,r);  
 }  
 if(e.c == '/'){  
 r.n= x2/x1;  
 Push(sum,r);  
 }  
 if(e.c == '^'){  
 r.n= pow(x2,x1);  
 Push(sum,r);  
 }  
 return ;  
}

void suma(pila\* sum,pila\* inver, float\* valores){  
 elemento e;  
 if(!Empty(inver)){  
 e=Pop(inver);  
 if(e.c > 64 && e.c < 92){  
 Push(sum,e);  
 suma(sum,inver,valores);  
 }  
 else{  
 eleccion(e,sum);  
 printf("\nValor de la operacion: %f\n",Top(sum).n);  
 suma(sum,inver,valores);  
 }  
 }  
else{  
 e=Pop(sum);  
 printf("\n\nResultado: %f",e.n);  
}  
return ;  
}

int regresaValor(char c){  
 int k=0;  
 if(c == '^')  
 k=3;  
 else{  
 if(c == '\*' || c == '/')  
 k=2;  
 else{   
 if(c == '+' || c == '-')  
 k=1;  
 }  
 }   
return k;  
}

float\* posfijo(pila\* pi,char expre[], pila\* guarda){   
 elemento e;  
 int n;  
 printf("\nExpresion en posfijo:\n");  
 float\* valores;  
 valores=(float\*)malloc(26\*sizeof(float));  
 int j;  
 for(j=0;j<26;j++)  
 valores[j]=-1;  
 int i;  
 for(i =0; i< strlen(expre); i++){  
 if(expre[i] == '(' ||expre[i] == '^'||expre[i] == '\*' || expre[i] == '/'||expre[i] == '+' || expre[i] == '-'){  
 if(!Empty(pi) && expre[i] != '('){  
 if(regresaValor(Top(pi).c) >= regresaValor(expre[i]))  
 cambio(expre[i],pi,guarda);  
 else{  
 e.c=expre[i];  
 Push(pi,e);  
 }  
 }  
 else{  
 e.c=expre[i];  
 Push(pi,e);  
 }  
 }  
 else{  
 if(expre[i] == ')'){   
 (Top(pi).c != '('){  
 e=Pop(pi);  
 printf("%c",e.c);  
 Push(guarda,e);  
 }  
 e=Pop(pi);  
 }  
 else{  
 e.c=expre[i];  
 printf("%c",expre[i]);  
 Push(guarda,e);  
 valores[expre[i]-65]=0;   
 }  
 }  
 }  
 if(!Empty(pi)){  
 while(!Empty(pi)){  
 e=Pop(pi);  
 printf("%c",e.c);  
 Push(guarda,e);  
 }  
 }  
 int k;  
 while(n != 1 || n != 2){  
 printf("\n\nEvaluar expresion?");  
 printf("\n1.- Si 2.- No (Salir)\n");  
 scanf("%i",&n);  
 if(n==1){  
 for(k =0; k< 26;k++){  
 if(valores[k] != -1){  
 printf("\nIntroduzca el valor de %c: ",k+65);  
 scanf("%f",&valores[k]);  
 }  
 }  
 printf("\n");  
 return valores;  
 }  
 else if (n == 2)  
 exit(1);  
 }  
}

void iniciarTodo(pila\* s,pila\* pi,pila\* guarda,pila\* inver,pila\* sum){  
 Initialize(s);   
 Initialize(pi);   
 Initialize(guarda);   
 Initialize(inver);   
 Initialize(sum);   
 return ;  
}

void destruirTodo(pila\* s,pila\* pi,pila\* guarda,pila\* inver,pila\* sum){  
 Destroy(sum);  
 Destroy(inver);  
 Destroy(guarda);  
 Destroy(pi);  
 Destroy(s);  
 return ;  
}

int main() {  
 pila s,pi,guarda,inver,sum;  
 float\* valores;  
 char expresion[100];  
 iniciarTodo(&s,&pi,&guarda,&inver,&sum);  
 printf("Introduce la cadena: ");  
 scanf("%s",&expresion);  
 printf("\n%s",expresion);  
 checar(&s,expresion);   
 printf("\n");  
 valores=posfijo(&pi,expresion,&guarda);  
 orden(&guarda,&inver,valores);  
 suma(&sum,&inver,valores);   
 destruirTodo(&s,&pi,&guarda,&inver,&sum);  
 free(valores);  
 return (EXIT\_SUCCESS);  
}

**Bibliografía (En formato IEEE)**

[1]L. Joyanes, *Estructura de Datos en C++*, 1st ed. España: McGraw-Hill España, 2011.

[2]S. Baase and A. Van Gelder, *Computer algorithms*, 1st ed. Delhi: Pearson Education, 2009.

[3]T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to algorithms*, 3rd ed. London, England: The MIT Press, 2009.